Chapter 18
Developing a Framework for the Effective Use of Learning Analytics:
A UK Perspective

Collette Gavan
Edge Hill University, UK

ABSTRACT

Research and experimentation is uncovering forms of best practice and possible factors on which to centre the analysis of students in an effective way, however learning analytics has yet to be comprehensively implemented country-wide in the United Kingdom. The chapter explores the current impact of learning analytics in higher education at mome discusses and observes the current vacancies with which a framework enabled to function with data visualisation could be utilised. The deliverable seeks to design an initial framework that has the potential to be utilised in a higher education setting for more effective and insightful decision making with regards to learner retention and engagement. This framework will combine the theory and scientific action of predictive analytics with a comparison of the most suitable data visualisation toolsets that are currently available in open-source software.

INTRODUCTION

Traditional approaches to teaching programming have encountered a number of issues. However when coupled with changes in both the demographic and political constitution of the student body it is clear that the strategies adopted to attain successful outcomes in relation to creating lifelong students need to re-evaluated. Clearly significant progress has been achieved, yet addressing the changing vision of higher education provision in relation to the Browne Report and the generational shift in technological demand places additional pedagogical requirements when designing implementing courses and assessment. This research investigates the extent to which the practical application of robotics affects undergraduate computing students' engagement in learning the Java programming language. The findings from this study have been interpreted using learning and academic analytics to derive meaning from the data gathered.

DOI: 10.4018/978-1-5225-2584-4.ch018

TEACHING PROGRAMMING TO UNDERGRADUATE STUDENTS

Teaching undergraduate students how to write code and develop their own programs has long proved a difficult task [Hagan and Markham, 2000; Kelleher and Pausch, 2005]. There are numerous hurdles that need to be overcome by the students in order to fully understand the concepts that underpin software development. Whilst syntax and semantics may be cited as issues [Hagan and Markham, 2000], these are specific details related to programming languages. It could be argued that the greatest challenge which face students when learning to program is the abstract nature involved in the processes of the software development cycle [Kölling, M et al, 2003]. An excellent example of this has been cited in the current trends to adopt object-oriented (OO) programming languages, such as Java and C++, as teaching languages [Sheard and Hagan, 1997; Xiaohui & Yaguan, 2010].

Whilst the selection of languages such as these can be justified by their widespread usage within the computing industry, thereby assisting the students in developing skills for employment following completion of their programmes of study, a significant issue lies in the theoretical underpinning for designing software using OO concepts. Within OO the classes, and related attributes and methods, are highly abstract. When learning to program using these languages students are not only facing the idiosyncrasies of the adopted programming language, but also the complexities of developing models of a software system which may or may not represent tangible objects in the real world [Azemi, 1995; Enzai et al, 2011; Sobrinho & Goncalves, 2011].

Addressing the issues faced by the students is hugely significant. Without the development of strategies and techniques to engage and assist students with their initial problems, then the students will soon become anxious and demotivated [Chan & Lee, 2005] and deem that programming is “too hard” or “not for them”. However, computer programming can be a hugely creative and innovative subject. The importance of developing creativity within the teaching environment as a means of enhancing student engagement has been the focus of previous studies, with a positive impact being identified [Enzai, 2011; Gibson, 2011; Rinkevich, 2011; Sun, 2011]. Indeed computer programming can be considered as part art-form, and there have been successful efforts to engage traditional non-programmers, such as artists and musicians, with the development of computing technology through subjects such as physical computing [O’Sullivan &Igoe, 2004; Igoe, 2007].

The pedagogic approach to teaching programming must therefore be considered with care. Often there is a misalignment between the goals of materials and the approach that is taken [Fincher, 1999]. Alternative approaches which align with working practices in the industry have been reported, such as those that align with a cyclic approach to developing skills [Barnes et al, 1997] or utilise concepts such as peer programming [Nagappan et al, 2003]. Equally attempts to instil more reflective practices have been reported to achieve greater engagement with the subject [Lilley and Rollins, 1995][Sitthiworachart and Joy, 2008].

The different student behaviours in confronting a problematic situation can be recognized [Greyling et al, 2006]. Perkins et al (1989) named two main types: stoppers and movers. In problematic situations stoppers simply stop and abandon all hope of solving the problem on their own, while movers keep trying, modifying their code and use feedback about errors effectively. There are also extreme movers, “tinkerers”, who cannot track their program, make changes more or less randomly, and like stoppers do not progress very much in their task.